Pattern occurrences Pvalues, Hidden Markov Models and Overlap Graphs

Mireille Régnier [*1], Evgenia Furletova[*2,3] , Mikhail Roytberg[2,4,5] and Victor Yakovlev[2]

[1] INRIA team AMIB, LIX and LRI-UPSud, 1 rue d'Estienne d'Orves, 91 120 Palaiseau, France
[2] Institute of Mathematical Problems of Biology, 142290, Institutskaya, 4, Pushchino, Russia
[3] Pushchino State University, 142290, Prospect Nauki, 5, Pushchino, Russia
[4] Laboratoire J.-V. Poncelet (UMI 2615), 119002, Bolshoy Vlasyevskiy Pereulok, 11, Moscow, Russia
[5] National Research University "Higher School of Economics", 101978, Myasnitskaya str., 20, Moscow, Russia

Email: Mireille Régnier *- mireille.regnier@inria.fr; Evgenia Furletova *- furletova@lpm.org.ru; Mikhail Roytberg - mroytberg@lpm.org.ru; Victor Yakovlev - v.yacovlev@gmail.com;

*Corresponding author

Abstract

Background: Finding new functional fragments in biological sequences is a challenging problem. Methods addressing this problem commonly search for clusters of pattern occurrences that are statistically significant. A measure of statistical significance is the P-value of a number of pattern occurences, i.e. the probability to find at least S occurrences of words from a pattern $\mathcal{H}$ in a random text of length N generated according to a given probability model.

Results: We present a novel algorithm SufPref computing an exact P-value for Hidden Markov model (HMM). The algorithm inductively traverses specific data structure - overlap graph ($OvGraph$). Nodes of the graph are associated with the overlaps of words from $\mathcal{H}$. Edges are associated to the prefix and suffix relations between ovelaps. An originality of our data structure is that pattern $\mathcal{H}$ need not be explicitely represented in nodes or leaves. The algorithm relies on the Cartesian product of the overlap graph and the graph of HMM states; the approach is analogous to the automaton approach from [1]. The gain in size of SufPref data structure leads to significant space and time complexity improvements. We suppose that all words in the pattern $\mathcal{H}$ are of the same length m. The algorithm SufPref was implemented as a C++ program; it can be used both as Web-server and a stand alone program for Linux and Windows; the program is available at http://lpm.org.ru/biosymbol/.

Background

Recognition of functionally significant fragments in biological sequences is a key issue in computational biology. Many functionally significant fragments are characterized by a set of specific words that is called a pattern and denoted $\mathcal{H}$ below. Patterns represent different biological objects, such as transcription factor binding sites [2–4], polyadenylation signals [5], protein domains, etc... Functional fragments recognition problem can be solved by finding sequences in which the words from a given pattern are overrepresented. Defining a meaningful significance criteria for this overrepresentation is a delicate goal, that, in turn, requires a clarification of the probability model. A current criteria is the so-called P-value computed as the probability that a random sequence of length N contains at least S occurrences of a pattern. There are a lot of methods for P-value computation designed for Bernoulli or Markov models. However, Hidden Markov models (HMM) were considered in few papers only [6, 7] despite the models are widely used in bioinformatics. This is a motivation to develop methods for P-value calculation with respect to HMMs. Existing methods for P-value calculation can be divided into several groups, reviews of the methods can be found in [8–10]. Studies on word probabilities started as early as Eighties with the seed paper [11] that introduces basic word combinatorics and derives inductive equations for a single word and a uniform Bernoulli model. Some works in the same vein, reviewed in [12] follow for several words, multi-occurrences and/or extended probability models. The time complexity is proportional to the text length N and the desired number of occurrences S : computations are carried out by induction for n ranging over $1 \cdots N$ and, for a given n, by induction on the number of occurrences. Although these "mathematics-driven" approaches allow for mathematical formula derivation, actual computation suffers from a combinatorial explosion when $|\mathcal{H}|$ or Markov order increase.

Later on, a first group of methods [13–17] formalizes systematically these inductions by the introduction of bivariate generating functions. Coefficients are the P-values to be computed. Expectations and variances for the number of occurrences of the different words in pattern $\mathcal{H}$ can be expressed explicitly in terms of these generating functions [14, 15, 18]. Moreover, coefficients may be computed from the analytical expression, when it is available, or through a suitable manipulation of a functional equation, where the

theoretical time complexity reduces to $S \log N$. Nevertheless, computing the generating function, or the functional equation, requires the computation of a system of equations or, equivalently, the determinant of a matrix of polynomials of size $O(|\mathcal{H}|)$. It takes $O(|\mathcal{H}|^3)$ operations and it is the main drawback of this approach.

A second group consists of asymptotic methods. They rely on convergence results to the normal law proved by [19] or [20]. An approximated P-value is derived, based on on Gaussian approximations [21] or Poisson approximations [22–25]. Nevertheless, this approximation is not suitable for exceptional words, when observed number of occurrences p significantly differs from the expected. This was proved experimentally by [26] or theoretically [27]. Large deviation principles are used in [28, 29] with a much better precision. Nevertheless, no computable formula are available for large sets.

A third group of methods revisits recursive P-value computation, with a $O(S \times N)$ time complexity. They avoid combinatorial explosion by a suitable use of appropriate data structures, tightly related to word overlap properties. Therefore, loss in time dependency to N or S is compensated by a gain on data structure size. A significant part of algorithms in this group are based on traversals of a specific graph. The graph may not be defined explicitly [30]. It can be based on the graph corresponding to the finite automaton recognizing the given pattern, see algorithms AhoPro [31], Spatt [25, 32] and Regexpcount [17]. MotifRank [33] that is designed for first order Markov models makes use of suffix sets. In [25, 32], a Markov chain embedding technique was suggested. Counting occurrences of regular patterns in random strings produced by Markov chains reduces to problems regarding the behavior of a first-order homogeneous Markov chain in the state space of a suitable deterministic finite automaton (DFA). In a recent paper [7], a probabilistic arithmetic automaton for computing P-values for a HMM was proposed. In this paper two algorithms were suggested. The first one has a time complexity $O(|Q|^2 \times N \times S \times |\Omega| \times |A|)$ and a space complexity $O(|Q| \times S \times |\Omega|)$, where $|Q|$ is the number of states of the HMM, $|\Omega|$ is the number of states of the automaton recognizing the given pattern, $|A|$ is the alphabet size. The second algorithm has a time complexity $O(|Q|^3 \times log(N) \times S^2 \times |\Omega|^3)$ and a space complexity $O(|Q|^2 \times S \times |\Omega|^2)$. This algorithm uses the "divide and conquer" technique. The drawback is the lack of control on the number of states $|\Omega|$ when $|\mathcal{H}|$ increases. Finally, despite these great efforts, existing methods perfom badly for rather big patterns. Besides this, most of the proposed algorithms are not implemented or implemented only for Bernoulli model or Markov models of small orders. Therefore, pattern occurrences probability problem is open.

The present paper provides an algorithm supporting HMM probability model. It assumes that all words

have the same length m and that a HMM with $|Q|$ states is given. It is a generalization of algorithm SufPref designed in [34] for Bernoulli models and Markov models of order K. It relies on recurrent equations based on overlap graph, whose vertices are associated with the overlaps of words from $\mathcal{H}$, and edges correspond to the prefix and suffix relations between overlaps. Time complexity is $O(|Q|^2 \times N \times S \times (|OV(\mathcal{H})| + |\mathcal{H}|))$ and space complexity is $(|Q|^2 \times (|OV(\mathcal{H})| + |\mathcal{H}|) + |Q| \times S \times m \times |OV(H)| + m \times |H|)$. In the case of a Markov model of order K, where $K \leq m$, bounds above can be reduced to $O(N \times S \times (K \times |A|^{K+1} + |OV(\mathcal{H})| + |\mathcal{H}|))$ for time and to $O(S \times K \times |A|^{K+1} + S \times m \times |OV(\mathcal{H})| + m \times |\mathcal{H}|)$ for space. Algorithm SufPref is implemented as a Web-server, see http://lpm.org.ru/biosymbol, and a stand-alone program for Windows and Linux. The program is available by request from the authors.

The paper is organized as follows. Basic notions on word overlaps are introduced, that lead to an overlap graph that is the main data structure to be used. Then, one recalls Hidden Markov models definition, and a probabilistic automaton is defined. Main text sets are defined and equations for their probabilities are derived. Next section describes the algorithm SufPref that computes these equations using the overlap graph as a main data structure. Finally, space and time complexity are analysed and our algorithm is compared with other methods [4, 24, 31, 35, 36].

Overlap words

Our approach strongly relies on overlaps of words from a given pattern. In this section we provide necessary definitions for these overlaps, following [34] notations.

Definition 1 *Given a pattern $\mathcal{H}$ over an alphabet* V*, a word w is an* overlap (*an overlap word*) *for $\mathcal{H}$ if there exist words* H *and* F *in $\mathcal{H}$ such as w is a proper suffix of* H *and w is a proper prefix of F. The set of overlaps of the pattern $\mathcal{H}$ is denoted $OV(\mathcal{H})$.*

Example: Let $\mathcal{H}$ be the set

$$\begin{aligned} \mathcal{H} \quad = \quad & \{ACATATA, ATACACA, ATACACA, ATAGATA, \\ & CATTATA, CTTTCAC, CTTTCCA, TACCACA\} \ . \end{aligned}$$

Overlap set is

$$OV(\mathcal{H}) = \{\epsilon, A, C, AC, CA, TA, ACA, ATA\} \ .$$

Notation: For a word w in $OV(\mathcal{H})$, one denotes

$$\mathcal{H}(w) = \{\mathrm{H} \in \mathcal{H} \mid \mathrm{H} \text{ ends with } w\} \ ,$$

with the convention $\mathcal{H}(\epsilon) = \mathcal{H}$.

Notation: $v' \sqsubseteq v$ ($v' \subset v$) means that v' is a suffix (proper suffix) of v; $v' \preceq v$ ($v' \prec v$) means that v' is a prefix (proper prefix) of v. The elements of $OV(\mathcal{H})$ that are proper prefixes (respectively suffixes) of a given word are totally ordered. The empty string is the minimal element. The maximal elements are crucial for our algorithms and data structures.

Definition 2 *Given a word w in $\mathcal{H} \cup OV(\mathcal{H}) \setminus \{\epsilon\}$, one denotes*

$$\begin{aligned} lpred(w) &= \max\{x | x \in OV(\mathcal{H}) \text{ and } x \prec w\} \ ; \\ rpred(w) &= \max\{x | x \in OV(\mathcal{H}) \text{ and } x \subset w\} \ . \end{aligned}$$

Two words H *and* F *from the pattern $\mathcal{H}$ are* equivalent *if they satisfy*

$$\begin{aligned} lpred(\mathrm{H}) &= lpred(\mathrm{F}) \ , \\ rpred(\mathrm{H}) &= rpred(\mathrm{F}) \ . \end{aligned}$$

Notation: Given two words x and w in $OV(\mathcal{H})$, let $\mathrm{H}^*(x, w)$ denote the equivalence class consisting of all words $H \in \mathcal{H}$ such that $lpred(H) = x$ and $rpred(H) = w$. Let $\mathcal{P}(\mathcal{H})$ denote the set of all equivalence classes on $\mathcal{H}$.

Definition 3 *An overlap $w \in OV(\mathcal{H})$ is called a* left deep node*, respectively a* right deep node*, if there exists a word* $\mathrm{H} \in \mathcal{H}$ *such that $w = lpred(\mathrm{H})$, respectively $w = rpred(\mathrm{H})$. The sets of all left and right deep nodes are denoted by $DLOV(\mathcal{H})$ and $DROV(\mathcal{H})$.*

Order relations are commonly associated to *oriented graphs*.

Notation: For a right deep node $r \in DROV(\mathcal{H})$, one denotes

$$\widetilde{\mathcal{H}}(r) = \{\mathrm{H} \in \mathcal{H} \mid r = rpred(\mathrm{H})\} \ .$$

Definition 4 *The overlap graph of a given pattern $\mathcal{H}$ is an oriented graph where the set of nodes is $OV(\mathcal{H})$ and the set of edges, $E(\mathcal{H})$, contains the* left, right *and* deep *edges, that are defined as follows:*

- *A* left edge *links node* s *to node* t *iff* $s = lpred(t)$*;*
- *A* right edge *links node* s *to node* t *iff* $s = rpred(t)$*;*
- *A* deep edge *links node* s *to node* t *iff exists a non-empty class* $\mathrm{H}^*(s,t)$ *in* $\mathcal{P}(\mathcal{H})$*.*

It is denoted OvGraph.

Definition 5 *Let* w *be in* $(OV(\mathcal{H}) \cup \mathcal{H}) \setminus \epsilon$*.*
The set of non-empty prefixes of w *that belong to* $OV(\mathcal{H})$ *is noted* $OverlapPrefix(w)$*. For any prefix* x *in* $OverlapPrefix(w)$*, let* $Back(x,w)$ *denote the suffix of* w *that satisfies the equation*

$$w = x \cdot Back(x,w) \ .$$

Let $Back(w)$ *denote* $Back(lpred(w), w)$*.*
Also for $\mathrm{H}^*(x,w) \in \mathcal{P}(\mathcal{H})$ *we note*

$$Back(\mathrm{H}^*(x,w)) = \bigcup_{\mathrm{H} \in \mathrm{H}^*(x,w)} Back(\mathrm{H}) \ .$$

Remark: One can ascribe to each deep edge (s,t) the class $\mathrm{H}^*(s,t)$ and to each left edge $(lpred(w), w)$ a word label $Back(w)$.

Probability models

We suppose that the probability distribution is described by a Hidden Markov Model (HMM). In this section, we recall some basic notions about HMMs and introduce the needed notations. In fact, it is shown in [7] that our definition is equivalent to the classical definition of HMM [37].

Definition 6 *A HMM* G *is a triple* $G =< Q, q_0, \pi >$*, where* Q *is the set of states,* $q_0 \in Q$ *is an initial state, and* π *is a function:* $Q \times \mathrm{V} \times Q \to [0,1]$ *such that* $\pi(\tilde{q}, a, q)$ *is the probability, being in state* $\tilde{q}$*, to generate symbol* a *and traverse to state* q*. For any state* $\tilde{q}$ *in* Q*, the function* π *meets the condition:*

$$\sum_{a \in \mathrm{V}} \sum_{q \in Q} \pi(\tilde{q}, a, q) = 1 \ . \tag{1}$$

A HMM G is called *deterministic* if for any $(\tilde{q}, a)$ in $Q \times \mathrm{V}$ there is only one state q such that $\pi(\tilde{q}, a, q) > 0$. In this case the function π can be described with two functions:

1. a transition function $\phi : Q \times \mathrm{V} \to Q$;

2. a probability function $\rho : Q \times \mathrm{V} \to [0, 1]$.

Namely, $\phi(\tilde{q}, a)$ is equal to the unique state q such that $\pi(\tilde{q}, a, q) > 0$ and $\rho(\tilde{q}, a)$ is $\pi(\tilde{q}, a, q)$.

A HMM $G =< Q, q_0, \pi >$ can be represented as a graph where Q is the set of vertices. Each edge is assigned with a label $a \in V$ and with a probability $p \in (0; 1]$. There exists an edge from $\tilde{q}$ to q with the label a and probability p iff $\pi(\tilde{q}, a, q) > 0$ and $p = \pi(\tilde{q}, a, q)$. The graph is called the traversal graph of HMM G.

Definition 7 *Let h be a path in the traversal graph of the HMM G. The label of h is the concatenation of the labels of edges that constitute the path h. The probability $Prob(h)$ of a path h is the product of the probabilities of the edges that constitute the path h.*

Definition 8 *The probability $Prob(w)$ of a word w with respect to the HMM G is the sum of probabilities of all paths that start in the initial state q_0 and have the label w.*
Let q and $\tilde{q}$ belong to Q and w be a word. By definition, the probability $Prob(\tilde{q}, w, q)$ to move from the state $\tilde{q}$ to the state q with the emmited word w is the sum of probabilities of all paths starting in the state $\tilde{q}$, ending in the state q and having the word label w.

To describe effective algorithms related to HMMs, we need the notion of reachability.

Definition 9 *Given a state $\tilde{q}$ and a string t, one notes*

$$ReachState(\tilde{q}, t) = \{q | Prob(\tilde{q}, t, q) \neq 0\} \ .$$

Given a state q and a string t, one notes

$$StartState(q, t) = \{\tilde{q} | Prob(\tilde{q}, t, q) \neq 0\} \ .$$

A state q is called t-reachable from state $\tilde{q}$ iff $Prob(\tilde{q}, t, q) \neq 0$.

Definition 10 *For a given word w, $AllState(w)$ is the set of states that are reached from initial state q_0 by at least one text with suffix w.*

Remark:

$$AllState(w) = \cup_{t \in V^*.w} ReachState(q_0, t) \ . \tag{2}$$

HMM and probabilistic automata

The definition of HMM is very close to the definition of probabilistic automaton PA, [38], see also the textbook [39]. The main difference is in the interpretation of the behavior of a model. For a HMM, one considers a label as a symbol emitted by the HMM; for automata, one imagines an automaton that proceeds a given word letter by letter. Another difference connected with the previous one is that PAs are typically used to describe word sets; thus, for a given PA, the subset of accepting states is defined. HMMs are mainly used to describe probability models and thus have no accepting states.

In applications, one often uses the probabilistic automata constructed as a Cartesian product of a deterministic automaton accepting a given set of words and an HMM describing the word probabilities, see e.g. [1, 7]. We use a similar construction in our work. In fact, we describe generalized probabilistic automata, GPA. As opposed to PAs, the edges in a graph that represents our automaton are labeled with words rather than with letters, and thus it can be named a generalized probabilistic automaton, analogously to the definition of generalized HMM [40].

An originality of SufPref is that words from pattern $\mathcal{H}$, or classes, that represent terminal states in classical automata need not be explicitely represented. Nevertheless, each class is uniquely associated to one deep edge.

Text sets

The computation of P-values will be done by induction on the text length n ($n = 1, \ldots, N$), and, for each given n, by induction on the number of occurrences s ($s = 1, ..., S$). It relies on specific sets of words introduced in a bit different form in [34], that by-turn was based on the ideas from [13], [12].

Definition 11 *Let $\mathcal{H}$ be a pattern.*

$$B(n, s) = \{T \in \mathrm{V}^n | T \textit{ contains at least } s \textit{ occurrences of the pattern } \mathcal{H}\} \ . \tag{3}$$

By convention, $B(n, 0) = V^n$.

Definition 12 *Given a deep right node $r \in DROV(\mathcal{H})$, one defines, for $s = 1, \ldots, S, S + 1$*

$$\begin{aligned} E(n, s, r) \quad = \quad & \{T \in \mathrm{V}^n | T \textit{ contains at least } s \textit{ occurences of } \mathcal{H} \ \& \\ & T \textit{ ends with } \mathrm{H} \in \mathcal{H}(r), \textit{ such that } rpred(\mathrm{H}) = r\} \ ; \end{aligned} \tag{4}$$

These sets are called E-sets.

Definition 13 *Let $w \in OV(\mathcal{H})$, one defines, for $s = 1, \ldots, S$*

$$\begin{aligned} R(n,s,w) &= \{T \in \mathrm{V}^n | T \textit{ contains exactly } s \textit{ occurences of } \mathcal{H} \ \& \\ &\quad T \textit{ ends with } \mathrm{H} \in \mathcal{H}(w)\} \ ; \end{aligned} \tag{5}$$

These sets are called R-sets.

Remark: Remark that

$$\begin{aligned} R(n,s,\epsilon) &= \{T \in \mathrm{V}^* | T \text{ contains exactly } s \text{ occurrences of } \mathcal{H} \ \& \\ &\quad T \text{ ends with } \mathrm{H} \in \mathcal{H}\} \ . \end{aligned}$$

Notation: Let r be a right deep node. We denote,

$$RE(n,s,r) = \{T \in R(n,s,r) | T \text{ ends with } \mathrm{H} \in \mathcal{H}(r) \text{ such that } r = rpred(H)\} \ . \tag{6}$$

Remark that

$$RE(n,s,r) = E(n,s,r) \setminus E(n,s+1,r) \ . \tag{7}$$

The following proposition gives the inductive relations allowing effective computation of probabilities of R-sets.

Proposition 1 *Let $w \in OV(\mathcal{H})$. If w is a deep right node, i.e. $w = rpred(\mathrm{H})$ for a word $\mathrm{H} \in \mathcal{H}$, then*

$$R(n,s,w) = RE(n,s,w) \bigcup \Big(\bigcup_{x \in OV(\mathcal{H}) : w = rpred(x)} R(n,s,x) \Big) \ , \tag{8}$$

otherwise,

$$R(n,s,w) = \bigcup_{x \in OV(\mathcal{H}) : w = rpred(x)} R(n,s,x) \ . \tag{9}$$

Proof follows from the definition of R-sets.

Remark: Set equations allow for a computation restricted to $OV(\mathcal{H})$ nodes. Additionnally, Equations (8) and (9) lead to the formulas expressing R-sets probabilities as a function of RE-sets probabilities in deep nodes.Therefore, the formulas allow for a computation on $OV(\mathcal{H})$ nodes and a memorization of R-sets probabilities in deep nodes. Further, (7) reduces computation of RE-sets probabilities to computation of probabilities of sets $E(n,s,r)$ where r is a right deep node.

Below we introduce D-sets and give the equations for D-sets, R-sets and E-sets leading to recursive equations for E-sets probabilities. The D-sets defined below consist of texts of length n containing at least s occurrences of the pattern $\mathcal{H}$, ending with a given non-empty overlap word w that has a common part with the last occurrence of the pattern $\mathcal{H}$.

Definition 14 *Let $w \in OV(\mathcal{H})$, $w \neq \epsilon$.*

$$\begin{aligned} D(n,s,w) &= \{T \in B(n,s) | w \textit{ is a suffix of } T \ \& \\ &\qquad s\textit{-th occurrence of the pattern } \mathcal{H} \textit{ intersects the suffix } w\} \ . \end{aligned} \tag{10}$$

By definition, $D(n,s,\epsilon) = \emptyset$.

Example: Consider the pattern $\mathcal{H} = \{ACACA, ATTAC\}$. Here $OV(\mathcal{H}) = \{\epsilon, A, AC, ACA\}$. The texts $t_1 = CTT\underline{ATT}\mathbf{ACA}$, $t_2 = TT\underline{AC}\overline{\underline{AC}\mathbf{ACA}}$ and $t_3 = CTAT\underline{AC}\mathbf{ACA}$ are in $D(9,1,ACA)$.
It means that all these words (1) are of length 9; (2) end with ACA; (3) have at least 1 occurrence of words from $\mathcal{H}$ (the occurrences are underlined or overlined) and (4) the 1st occurrence intersects the suffix ACA. Remark that t_2 contains 2 occurrences of $\mathcal{H}$ and all of them overlap with ACA. Therefore, t_2 belongs to $D(9,2,ACA)$. In contrast, $t_4 = TATT\underline{AC\mathbf{ACA}}$ does not belong to $D(9,1,ACA)$ because it's 1st occurrence of $\mathcal{H}$ does not intersects the suffix ACA. However, t_4 belongs to $D(9,2,ACA)$.

The next propositions describe the relation between D-sets and R-sets.

Proposition 2 *Let $w \in OV(\mathcal{H})$, $w \neq \epsilon$. Then*

$$D(n,s,w) = \cup_{x \in OverlapPrefix(w)} R(n - |Back(x,w)|, s, x) \cdot Back(x,w) \ . \tag{11}$$

Informally speaking, here x is the common part of the suffix w of the text t of length n and the prefix of t ending with the s-th occurrence of $\mathcal{H}$ in t. Remark that it follows from Definition 5 that : (1) ϵ is not in $OverlapPrefix(w)$, (2) w in $OverlapPrefix(w)$. The formal proof is given in Supplementary materials.
Notation: For a prefix $w \in OV(\mathcal{H})$ and any integer n, one denotes

$$k(n,w) = n - m + |w| \ . \tag{12}$$

Proposition 3 *Let $w \in OV(\mathcal{H}) \setminus \epsilon$, $n \geq m, s \geq 1$. Then*

$$D(k(n,w),s,w) = D(k(n,lpred(w)),s,lpred(w)) \cdot Back(w) + R(k(n,w),s,w) \ . \tag{13}$$

Proof follows from the proposition 2, see Supplementary materials.

Example: Consider the pattern $\mathcal{H}$ and texts t_1, t_2, t_3 from the previous example. Obviously, t_1, t_2 are in $D(8,1,AC).A$ and t_3 is in $R(9,1,ACA)$, where AC is $lpred(ACA)$ and A is $Back(ACA)$.

Corollary: If $lpred(w) = \epsilon$ then $D(n,s,w) = R(n,s,w)$.

One observes that, whenever $n < m$, $B(n,s) = \emptyset$, and for all $w \in OV(\mathcal{H})$ and $r \in DROV(\mathcal{H})$, $R(n,s,w) = E(n,s,r) = \emptyset$

Theorem 1 *Let $n \geq m$ and $r \in DROV(\mathcal{H})$.*

1. *Sets $B(n,s)$ and $E(n,s,r)$ meet following equations:*

$$B(n,s) = B(n-1,s)\cdot \mathrm{V} \quad \cup R(n,s,\epsilon) \tag{14}$$

$$E(n,1,r) = V^{n-m}\cdot \widetilde{\mathcal{H}}(r) \tag{15}$$

$$E(n,s+1,r) = (B(n-m,s)\cdot \widetilde{\mathcal{H}}(r)) \bigcup (\bigcup_{\mathrm{H}^*(w,r)\in \mathcal{P}(\mathcal{H})} D(k(n,w),s,w)\cdot Back(\mathrm{H}^*(w,r))) \tag{16}$$

2. *Unions (14) -(16) are disjoint, i.e. their terms have empty intersection.*

Notation: Given two integers n and s, and a class $\mathrm{H}^*(x,r)$, one introduces F-sets and C-sets as follows.

$$F(n,s+1,r) = B(n-m,s)\cdot \widetilde{\mathcal{H}}(r) \ ; \tag{17}$$

$$C(n,s+1,x,r) = D(k(n,x),s,x)\cdot Back(\mathrm{H}^*(x,r)) \ . \tag{18}$$

Remark: Being observed that a class is uniquely associated to a deep edge, formula (16) rewrites

$$E(n,s+1,r) = F(n,s+1,r) \bigcup (\bigcup_{x:\ (x,r) \text{ is deep edge}} C(n,s+1,x,r)) \ . \tag{19}$$

Proof:

1. Consider statement (14). A text t is in $B(n,s)$ iff either its prefix of length $n-1$ contains at least s occurences of $\mathcal{H}$ or a s-th occurrence H from $\mathcal{H}$ ends at position n. In the first case, t is in $B(n-m,s)\cdot \mathrm{V}$. In the second case, text t belongs to $R(n,s,\epsilon)$. The two cases are mutually exclusive; therefore $B(n,s)$ is a disjoint union and (14) is proved.

2. Statement (15) directly follows from the definition of $E(n, 1, r)$.

3. Consider statement (16). Let Y denote the right side of equation (16).

   (a) Firstly, we proof that $E(n, s+1, r) \subseteq Y$. When a text t is in $E(n, s+1, r)$, it ends with a word $\mathrm{H} \in \mathcal{H}$ such that $r = rpred(\mathrm{H})$. Let $w = lpred(\mathrm{H})$. This suffix H of t may either overlap with s-th occurrence of the pattern or not. In the latter case, t is in $B(n-m, s) \cdot \mathrm{H}$. The two cases are mutually exclusive. Consider the former case. Let x be an overlap between the suffix H of the text t and the s-th occurence of $\mathcal{H}$ in t. Obviously, $x \in OverlapPrefix(w)$. By definition of R-sets, $t \in R(k(n,x), s, x) \cdot Back(x, \mathrm{H})$. Observing that

   $$Back(x, \mathrm{H}) = Back(x, w) \cdot Back(\mathrm{H})$$

   we obtain

   $$t \in R(k(n,x), s, x) \cdot Back(x, w) \cdot Back(\mathrm{H}) \ .$$

   Note, $k(n,x) = k(n,w) - |Back(x,w)|$.

   According to the proposition 2,

   $$R(k(n,x), s, x) \cdot Back(x, w) \subseteq D(k(n,w), s, w) \ .$$

   Thus

   $$t \in D(k(n,w), s, w) \cdot Back(\mathrm{H}) \ .$$

   Note, if $\mathrm{H} \in \mathrm{H}^*(w, r)$ then $Back(\mathrm{H}) \subseteq Back(\mathrm{H}^*(w, r))$. Therefore,

   $$D(k(n,w), s, w) \cdot Back(\mathrm{H}) \subseteq D(k(n,w), s, w) \cdot Back(\mathrm{H}^*(w, r)).$$

   This yields that $t \in Y$.

   (b) Proof that $Y \subseteq E(n, s+1, r)$. Let $t \in Y$, i.e $t \in B(n-m, s) \cdot \widetilde{\mathcal{H}}(r)$ or $t \in D(k(n,w), s, w) \cdot Back(\mathrm{H}^*(w, r))$. By definitions of B and D-sets:

   - t has the length n;
   - t contains at least $s+1$ occurences of the pattern;
   - t ends with $\mathrm{H} \in \mathrm{H}^*(w, r)$.

   Thus $t \in E(n, s+1, r)$.

Remark: All unions in equations (14) - (16) are disjoint. Therefore the probability of the set in the left part of an equation is the sum of probabilities of sets in the right side.

Algorithms

Our goal is to compute $Prob(B(N,S))$, that is the probability to find at least S occurrences of a pattern $\mathcal{H}$ in a random text of length N, given a HMM $G =< Q, q_0, \pi >$. The algorithm SufPref, see Figure 1, computes the probability by induction on text length n, where $m \leq n \leq N$, and, for a given n, by induction on $s, 1 \leq s \leq S$.

The computation within the main loop is based on equations (7)- (9), (13)-(20), related to B-sets, R-sets, RE-sets, D-sets, C-sets, F-sets and E-sets. The equations for the probabilities of the sets are based on the following observations. First, all unions in the text equations are disjoint. Second, an item of a set union is a set with already known probability or concatenation of such sets. In the latter case the probability $Prob(q', L_1 \cdot L_2, q")$ can be computed by the formula

$$Prob(q', L_1 \cdot L_2, q") = \sum_{q \in Q} Prob(q', L_1, q) \cdot Prob(q, L_2, q") \ , \tag{20}$$

where $Prob(q', L, q)$ is a probability being in the state q' go to the state q emitting a word v from the set L. The computation related to texts of length n will be referred to as n-th stage of the algorithm's work. The main computation within n-th stage is done by traversal of $OvGraph$ following left and deep edges. Updating of auxiliary information stored in nodes of $OvGraph$ is performed by a bottom-up traversal of $OvGraph$ using right edges.

Computation on inductive equations, and some preprocessing, relies on a generic procedure, analogous to the *forward algorithm* for HMM [37], see also [6].

Preprocessing and data structures

On the preprocessing stage we initialize the global data structures of the algorithm, i. e. the $OvGraph$, including auxiliary structures assigned to its nodes and some other structures that are described at the end of this subsection.

Overlap Graph The graph $OvGraph$ is built from the Aho-Corasick trie $T_{\mathcal{H}}$ for the set $\mathcal{H}$ [41]. The nodes belonging to the $OvGraph$ correspond to the overlaps and therefore can be easily revealed using suffix links of the Aho-Corasick trie, see [34] and supplementary materials for details of the procedure. The nodes of $OvGraph$ are assigned with additional data (constant data and data to be updated at each stage $n = m+1, \ldots, N$). All these data are initialized at the preprocessing stage, see below.

Constant transition probabilities related to nodes of overlap graph During the computation, algorithm SufPref uses some probabilities that are constant and can be precomputed and stored.

Let w be an overlap word. We denote by $PriorState(w, q)$ the set of states $\tilde{q} \in AllState(lpred(w))$ such that q is $Back(w)$-reachable from $\tilde{q}$, i.e.

$$PriorState(w, q) = AllState(lpred(w)) \cap StartState(q, Back(w)) \ .$$

For each deep edge (x, r) and its associated class $\mathrm{H}^*(x, r)$, one notes

$$PriorState(\mathrm{H}^*(x, r)), q) = AllState(x) \cap [\cup_{\mathrm{H} \in \mathrm{H}^*(x,r)} StartState(q, Back(\mathrm{H}))] \ .$$

- For each node w and all states q in $AllState(w)$ and $\tilde{q}$ in $PriorState(w, q)$, we store the "left transition probability" $Prob(\tilde{q}, Back(w), q)$. The left transition probabilities are used for the computation of D-sets probabilities, see (14);

- Given a right deep node r, we store, for each class $\mathrm{H}^*(x, r)$, the "deep transition probabilities" $Prob(\tilde{q}, Back(\mathrm{H}^*(x, r)), q)$ where q ranges over $AllState(\mathrm{H}^*(x, r))$ and $\tilde{q}$ ranges over $PriorState(\mathrm{H}^*(x, r), q))$. The probabilities are needed for the computation of E-sets probabilities, see (16);

- Given a right deep node r, the "word probabilities" $Prob(\tilde{q}, \widetilde{\mathcal{H}}(r), q)$ are memorized for states q in $AllState(r)$ and $\tilde{q}$ in Q. They are used to compute probability of the set $B(n - m, s) \cdot \widetilde{\mathcal{H}}(r)$, see (16).

The sets of states $AllState(w)$ and $PriorState(w)$, left and deep transition probabilities and word probabilities are computed in a depth-first traversal along left edges of $OvGraph$, see details in Supplementary materials.

Updatable Probabilities related to nodes of overlap graph At the beginning of n-th stage, for each pair $< w, q >$, where $w \in OV(\mathcal{H})$ and $q \in AllState(w)$ we store a $(m - |w|) \times S$ matrix with R-sets probabilities $Prob(R(l, s, w), q)$, where $l \in [k(n, w), n - 1], s = 1, \ldots, S$. The probabilities are updated at the end of the n-th stage.

At the preprocessing stage, we compute the probabilities for $n = 1, \ldots, m$, $s = 1, \ldots, S$ and $q \in AllState(w)$ according to the formulas:

$$Prob(R(m, 1, w), q) = Prob(\mathcal{H}(w), q);$$

if $n < m$ or ($n = m$ and $s > 1$),

$$Prob(R(n, s, w), q) = 0.$$

The global data unrelated to overlap graph Besides the data related to nodes of $OvGraph$ we store the following data.

- Transition probabilities. For each $\tilde{q}, q \in Q$ we store constant probability

$$TransProb(\tilde{q}, q) = \sum_{a \in \mathrm{V}} \pi(\tilde{q}, a, q) \ ;$$

At the beginning of n-th stage, we store the following values

- For each $q \in Q$, updatable probabilities $Prob(V^{n-m-1}, q)$. It is used for computation of $E(n, 1, r)$ by the formula (15);

- For each $s = 1, \ldots, S$ and $q \in Q$, updatable B-sets probabilities $Prob(B(n - m - 1, s), q)$. At the preprocessing stage, we compute the probabilities for $n = 1, \ldots, m$, $s = 1, \ldots, S$ and $q \in Q$ according to the formulas:

$$Prob(B(m, 1), q) = Prob(\mathcal{H}, q) \ ;$$

if $n < m$ or ($n = m$ and $s > 1$),

$$Prob(B(n, s), q) = 0.$$

Main loop

The aim of the n-th stage (see procedure $ComputeMainLoop(n)$, Figure 2) is to compute for all $s = 1, \ldots, S$ (internal loop, see lines 7-43) the values

- $Prob(B(n - m, s), q)$, $n > 2m$;

- $Prob(R(n, s, w), q)$ for all $w \in OV(\mathcal{H})$, $q \in AllState(w)$.

At the preliminary step we initialize local arrays $EProb(r)$ and $EProbPrev(r)$ assigned to each deep right node r; all arrays are of length $|Q|$, see below. The internal loop, lines 7-43, consists of three parts, below the value s is fixed.

At part A, the values $Prob(B(n - m, s), q)$ are computed according to the formula (14); the values $Prob(B(n - m - 1, s), q)$ and $Prob(R(n - m, s, \epsilon), q)$ were computed and stored at the previous stages.

The aim of part B of $ComputeMainLoop(n)$ is to compute the values $Prob(E(n, s+1, r), q)$ for all $r \in DROV(\mathcal{H})$, $q \in AllState(r)$ following (15) - (20).

The computation is done by recursive depth-first traversal of $OvGraph$ using left edges. Firstly, for all visited nodes $w \in OV(\mathcal{H}) \setminus \epsilon$ and states $q \in AllState(w)$ the procedure computes $Prob(D(k(n, w), s, w), q)$ by the formula (13). To make the computations by the formula (13) one needs the values $Prob(D(k(n, lpred(w)), s, lpred(w)), \tilde{q})$. For this reason, when a node w is visited, the procedure stores local arrays of size $|Q|$ with $Prob(D(k(n, x), s, x), \tilde{q})$, where $x \in OverlapPrefix(w)$, $\tilde{q} \in AllState(x)$. When a node w is visited this information is available for any x in $OverlapPrefix(w)$.

If w is a left deep node, then for all deep edges (w, r) the procedure computes $Prob(C(n, s+1, w, r), q)$ for all $q \in AllState(r)$ by the formula (18). If w is a right deep node then the procedure computes $Prob(F(n, s+1, w), q)$ using formula (17). Thus, the values $Prob(E(n, s+1, r), q)$ are computed cumulatively in corresponding cells of $EProb(r)[q]$ according (19), see lines 21, 28. Thus, at the end of part B, $EProbPrev(w)[q] = Prob(E(n, s, w), q)$ and $EProb(w)[q] = Prob(E(n, s+1, w), q)$.

At part C, the values $Prob(R(n, s, w), q)$ are computed according to the formulas (8), (9). For a right deep node r, the procedure computes first (see line 36)

$$Prob(RE(n, s, r), q) = EProbPrev(w)[q] - EProb(w)[q],$$

then it prepares arrays $EProb(r)$ and $EProbPrev(r)$ for the computation with the next value of s, lines 37, 38. The values $Prob(R(n, s, w), q)$ for various nodes w are computed according (8), (9), see line 40.

Remark: The above traversal is implemented with a recursive procedure with starting call at the root of $OvGraph$. Therefore right edges go from root to leaves.

Post-processing

At the post-processing step of the algorithm (see Figure 1, line 10), P-value $Prob(B(N, S))$ follows by summation over Q states:

$$Prob(B(N, S)) = \sum_{q \in Q} Prob(B(N, S), q) \ .$$

To improve implementation of the algorithm SufPref we slightly modified the algorithm, see Supplementary materials. In the version of SufPref described in the paper, the $OvGraph$ traversals are performed inside the internal loop on number of occurrences, see Fig. 2 of the paper. In opposite, in the modified version of the algorithm the internal loop is performed during processing of a node of $OvGraph$

within traversals of the graph. This modification reduces the number of recursive calls of graph traversal procedures that, in turn, allows to save running time.

To simplify the presentation, we omitted some details of the implemented algorithm. The detailed description of the algorithm is given in the Supplementary materials.

Discussion

Space complexity depends on input data, temporary data used at the preprocessing step, the main data structure *OvGraph* and the working data unrelated to the *OvGraph*. The space complexity is mainly determined by the memory needed for the data related to the *OvGraph* and temporary data used at the preprocessing step. Thus we first briefly consider data unrelated to overlap graph, then consider *OvGraph* data. Input data consist of text length N, number of occurrences S, representations of an HMM and a pattern $\mathcal{H}$. The data related to the pattern representation are included into the data related to *OvGraph* nodes and will be considered below. Storage size for an HMM is $O(|Q|^2 \times |\mathrm{V}|)$. Thus the input data size is $O(|Q|^2 \times |\mathrm{V}|)$.

At the preprocessing stage the algorithm uses a temporary structure (Aho-Corasick trie) to build the *OvGraph*. The memory needed for Aho-Corasick trie is $O(m \times |\mathcal{H}|)$, m is the pattern length. The memory is released step by step within the construction of *OvGraph*, thus total memory used during the construction of *OvGraph* is $O(m \times |\mathcal{H}|)$.

The data unrelated to *OvGraph* consist of B-sets probabilities $Prob(B(n-m-1,s),q)$ and probabilities $Prob(V^{n-m-1},q)$, $q \in Q$. Needed memory is $O(|Q| \times S)$ and $O(|Q|)$ correspondingly. Within the main loop we use local array with D-sets probabilities (at most m arrays) and arrays $EProbPrev(r)[], EProb(r)[]$ (for all $r \in DROV(\mathcal{H})$). All these arrays are of size $O(|Q|)$; and therefore needed memory to store all of the arrays is $O(|Q| \times m + |Q| \times |DROV(\mathcal{H})|)$. As we will see, all the memory, except memory needed to store Aho-Corasick trie, in total does not affect the space complexity of the algorithm.

Now consider the data related to the *OvGraph*. The *OvGraph* structure is determined by the pattern $\mathcal{H}$. The number of nodes and the number of left and right edges is $O(|OV(\mathcal{H})|)$ that is upper bounded by $m \times |\mathcal{H}|$. However, usually $|OV(\mathcal{H})| << m \times |\mathcal{H}|$, see Table 1. The number of deep edges is equal to the number of classes, $|\mathcal{P}(\mathcal{H})|$, that is upper bounded by $|\mathcal{H}|$. Then the storage size for *OvGraph* is $O(|\mathcal{H}| + |OV(\mathcal{H})|)$. The data assigned to the nodes of *OvGraph* can be divided into two groups, constant data and updatable data. The constant data consist of left transition probabilities assigned to nodes of the

OvGraph, deep transition probabilities assigned to the deep edges and word probabilities assigned to right deep nodes. The updatable data are probabilities of R-sets assigned to all nodes. The definitions are given in the section "text sets". More precisely, left transition probabilities $Prob(\tilde{q}, Back(w), q)$ are stored in the memory associated with the node w; deep transition probabilities $Prob(\tilde{q}, Back(\mathrm{H}^*(x,r), q)$ are stored in the memory associated with deep edge (x,r); word probabilities $Prob(\tilde{q}, \mathrm{H}(r), q)$ are stored in the memory associated with the right deep node r. As a whole, it gives

$$O(|Q|^2 \times |OV(\mathcal{H})|) + O(|Q|^2 \times |\mathcal{P}(\mathcal{H})|) + O(|Q|^2 \times |DROV(\mathcal{H})|) \leq O(|Q|^2 \times (|OV(\mathcal{H})| + |\mathcal{H}|)).$$

To store R-sets probabilities one needs $O(S \times |Q| \times m \times |OV(\mathcal{H})|)$ memory. Thus the size of memory needed to store global data related to *OvGraph* is

$$O(|Q|^2 \times (|OV(\mathcal{H})| + |\mathcal{H}|) + |Q| \times S \times m \times |OV(\mathcal{H})|) \ .$$

Finally, the overall space complexity of the algorithm is

$$O(|Q|^2 \times (|OV(\mathcal{H})| + |\mathcal{H}|) + |Q| \times S \times m \times |OV(\mathcal{H})| + m \times |\mathcal{H}|) \ .$$

Observe that grouping pattern words in deep nodes saves a $O(S \times |Q| \times m \times |\mathcal{P}(\mathcal{H})|)$ memory for R-sets.

Remark: Computer experiments show that $|\mathcal{P}(\mathcal{H})| \sim c \cdot |OV(\mathcal{H})|$, where $0 \leq c \leq 1$, see Table 1, Supplementary materials and [34] for details. For randomly generated patterns according to uniform Bernoulli model, $c \sim 1$. But for a majority of pattern described by Position-Specific Scoring Matrixes and cut-offs $c \leq 0.1$. Therefore, this implicit representation of patterns saves practically half of the total space.

Time complexity The algorithm $SufPref($ see figure 1) consists of three parts: preprocessing, main loop and post-processing. The time complexity of the pre-processing part is mainly determined by construction of Aho-Corasick trie, construction of *OvGraph* and their traversals. The complexity is $O(|Q|^2 \times m \times |\mathcal{H}|)$, see Supplementary materials for details. The time complexity of the post-processing part (see lines 5-10) is $O(m \times |Q|^2)$.

The time complexity of the algorithm $SufPref$ is mainly determined by the main loop (see lines 2-4), i.e. by total run-time of the computation $ComputeMainLoop(n)$ for $n = m+1, \ldots, N$. The computation of $ComputeMainLoop(n)$ for a given n consists of four parts: preliminary step, A, B and C, it is presented in Figure 2. The parts A, B and C run for S values of s. At the preliminary step (lines 1-5) one computes $Prob(E(n,1,r), q)$ for all $r \in DROV(\mathcal{H})$ and $q \in AllState(r)$, it requires $O(|Q|^2 \times |DROV(\mathcal{H})|)$ operations. Within the part A (lines 8-12), computing probabilities $Prob(B(n-m,s), q)$ for all $s = 1, \ldots, S$ and $q \in Q$

requires $O(S \times |Q|^2)$ operations. Analogously, to execute parts B and C (lines 13-31 and 32-42 respectively) one needs $O(S \times |Q|^2 \times (|\mathcal{H}| + |OV(\mathcal{H})|))$ and $O(S \times |Q| \times |OV(\mathcal{H})|)$ operations respectively. As a whole, $O(S \times |Q|^2 \times (|\mathcal{H}| + |OV(\mathcal{H})|))$ operations are needed to execute $ComputeMainLoop(n)$. Therefore, the time complexity of the algorithm $SufPref$ is

$$O(N \times S \times |Q|^2 \times (|\mathcal{H}| + |OV(\mathcal{H})|)) \ .$$

Further refinements Complexity results are presented with (possibly rough) upper bounds. In particular, $|Q|^2$ factor arises from transition probabilities representation. It actually stands for the sum of the cardinalities of $PriorState$ sets in a given node. In practical cases, this number may be significantly smaller than $|Q|^2$. In particular, this is the case for Markov models that can be treated as a special case of Hidden Markov Models. In the case of Markov models of order K for an overlap node w, such that $|w| \geq K$, only one state is associated with w. We use the technique of "reachable states", see section "Probability models" to take into account this issue. The technique does not allow decreasing the upper bounds in general case but leads to the significant improvement of the software. In the same time it the technique combined with careful processing of the pattern word set allows one to obtain better complexity bounds for the Markov case. Namely, $O(S \times m \times (K \times |\mathrm{V}|^{K+1} + |OV(\mathcal{H})|) + m \times |\mathcal{H}|)$ space complexity and $O(N \times S \times (K \times |\mathrm{V}|^{K+1} + |\mathcal{H}| + |OV(\mathcal{H})|))$ time complexities are achievable. The details of the special algorithm for the Markov case and proof of the above bounds will be presented in the separate paper.

Comparison with the existing algorithms Theoretical complexities analysis shows that SufPref is one of the best algorithms for P-value computation. The complexities of SufPref are compatible with complexities of algoritms based on finite automata. Superiority of algorithms depends on used data structures. Comparison of number nodes of $OvGraph$ and number of states of minimal automaton for a given pattern is given in the paper [34]. Also in the paper was shown that an average number of overlaps in random patterns generated according to Bernoulli models is proportional to the number of words in the patterns and is independent of the length of the words.

For Bernoulli and first order Markov model cases we have compared the program SufPref with the implementation of the program AhoPro [31]. The program AhoPro is one of the most efficient available programs computing exact P-value. We have calculated P-value with the following input parameters: (1) alphabet - $\{A, C, G, T\}$; (2) Bernoulli probabilities of letters - $\{0.25, 0.25, 0.25, 0.25\}$; Markov model is described by 4×4 matrix where all elements are 0.25; (3) text length -1000; (4) minimal number of occurrences -10 and (5) two types of patterns: random patterns of lengthes 8 and 12 and patterns of

lengthes 8 and 12 presented by Position-Specific Scoring Matrix (PSSM) and different cut-offs. Note, a pattern presented by PSSM and cut-off consists of all words which score according to PSSM is grater then cut-off. The matrices PSSM were created by Dan Pollard for Drosophila genes (http://www.danielpollard.com/matrices.html) and is given in Table 3. The results of the experiments for patterns of length 12 presented by PSSM are given in the Tables 1, 2. The results of other experiments are given in the Supplementary materials. In the tables, the number of nodes of Aho-Corasick trie (the size of automaton used by AhoPro) is denoted by NAC. The running time is given in seconds and size of used memory is given in megabytes.

It was shown that, for all considered case, our algorithm is faster than AhoPro by more than four and two times for Bernoulli and Markov models respectively. It outperforms AhoPro in space except for several pattern of small sizes.

Pattern $\mathcal{H}$	$\lvert\mathcal{H}\rvert$	$\lvert OV(\mathcal{H})\rvert$	$\lvert\mathcal{P}(\mathcal{H})\rvert$	NAC	P-value
PSSM(12,9)	280	16	144	704	2.13435871E-25
PSSM(12,8)	816	50	563	1725	9.78557008E-21
PSSM(12,7)	2056	89	1402	4917	9.29720887E-17
PSSM(12,6)	5272	183	3454	11325	1.01393226E-12
PSSM(12,5)	11600	261	6761	21469	2.14446331E-09
PSSM(12,4)	24216	553	16569	45677	1.88185558E-06
PSSM(12,3)	47448	987	35632	87341	0.00053964007
PSSM(12,2)	91432	1663	76447	157613	0.04556358352
PSSM(12,1)	170032	3563	153626	283237	0.54810104018
PSSM(12,0)	284488	7499	275084	474701	0.97468948572
PSSM(12,-1)	467056	14428	461442	766549	0.99997857117

Table 1: Common information about experiments. The patterns are of length 12, they are presented by matrix PSSM (see Table 3) and a series of cut-offs.

Conclusions

The work presents the approach to compute the P-value of multiple pattern occurrence within a randomly generated text of a given length. The approach provides significant space and time improvements compared to the existing software that is crucially important for applications. The improvements are achieved due to usage of an overlap graph; the nodes of the graph correspond to overlaps between pattern words. Taking into account overlaps between the pattern words allows one to decrease necessary space and time. Remqrk that the nodes of the extensively used structure Aho-Corasic trie, used in particular by the algorithm AhoPro, are associated with prefixes of pattern words. The number pf prefixes is much larger than the number of overlaps.

Experiments parameters				Time			Space		
Pattern $\mathcal{H}$	$\lvert\mathcal{H}\rvert$	$\lvert OV(\mathcal{H})\rvert$	Prob Distrib	SufPref	AhoPro	Aho/SP	SufPref	AhoPro	Aho/SP
PSSM(12,9)	280	16	Bernoulli	0.02	0.56	24.48	1.44	1.34	0.93
PSSM(12,8)	816	50	Bernoulli	0.08	1.39	17.59	1.54	1.85	1.20
PSSM(12,7)	2056	89	Bernoulli	0.19	4.00	21.39	1.83	3.63	1.98
PSSM(12,6)	5272	183	Bernoulli	0.46	9.47	20.58 ov	2.38	6.75	2.83
PSSM(12,5)	11600	261	Bernoulli	0.89	18.75	20.98	3.20	11.95	3.74
PSSM(12,4)	24216	553	Bernoulli	2.26	42.21	18.70	5.45	24.29	4.45
PSSM(12,3)	47448	987	Bernoulli	5.00	83.52	16.71	9.15	45.68	4.99
PSSM(12,2)	91432	1663	Bernoulli	10.90	15.99	1.47	15.26	81.52	5.34
PSSM(12,1)	170032	3563	Bernoulli	22.85	294.66	12.89	26.20	145.71	5.56
PSSM(12,0)	284488	7499	Bernoulli	44.19	529.89	11.99	42.91	243.68	5.68
PSSM(12,-1)	467056	14428	Bernoulli	73.90	896.74	12.13	68.34	392.95	5.75
PSSM(12,9)	280	16	Markov	0.04	0.58	14.74	1.48	1.38	0.94
PSSM(12,8)	816	50	Markov	0.12	1.41	11.46	1.58	1.90	1.20
PSSM(12,7)	2056	89	Markov	0.27	4.05	15.11	1.87	3.64	1.94
PSSM(12,6)	5272	183	Markov	0.63	9.57	15.22	2.44	6.80	2.79
PSSM(12,5)	11600	261	Markov	1.15	19.08	16.63	3.32	11.95	3.60
PSSM(12,4)	24216	553	Markov	2.79	42.62	15.25	5.48	24.33	4.44
PSSM(12,3)	47448	987	Markov	6.05	84.52	13.96	9.18	45.73	4.98
PSSM(12,2)	91432	1663	Markov	12.89	157.18	12.20	15.34	81.56	5.32
PSSM(12,1)	170032	3563	Markov	28.32	297.45	10.50	26.32	145.76	5.54
PSSM(12,0)	284488	7499	Markov	55.98	534.70	9.55	54.68	243.73	4.46
PSSM(12,-1)	467056	14428	Markov	101.54	904.54	8.91	68.76	392.74	5.71

Table 2: Comparison of running time and used memory sizes. The patterns are of length 12, they are presented by matrix PSSM (see Table 3) and a series of cut-offs. The running time is given in seconds and size of used memory is given in megabytes.

Another advantage of the described approach is that, unlike most existing algorithms and programs, it allows to deal with Hidden Markov Models, the most general class of popular probabilistic models. The algorithm relies on the Cartesian product of the overlap graph and the graph of HMM states; the approach is analogous to the automaton approach from [1]. We carefully analyze the structure of the Cartesian product, e.g. reachability of vertices that leads to extra improvement of time and space complexity. Despite that Bernoulli and Markov models can be treated as special HMMs we implemented specialized versions of software for these classes of models. The version of SufPref designed for Bernoulli models was presented in our previous paper [34]; the peculiarities related to Markov models of high orders will be presented in a separate paper.

The algorithm is implemented as an open software; it is available as programs for Windows and Linux families of operating systems and as Web-service. The implementation of the algorithm SufPref was compared with program AhoPro for Bernoulli model and first order Markov model. The comparison shows that our algorithm for all considered cases is faster than AhoPro in more than four times for

-	A	C	G	T
1	0.368	-2.197	-0.588	0.636
2	0.000	0.000	0.000	0.000
3	0.636	-2.197	-2.197	0.636
4	-2.197	-2.197	-2.197	1.299
5	1.299	-2.197	-2.197	-2.197
6	-2.197	-2.197	-2.197	1.299
7	-2.197	1.299	-2.197	-2.197
8	-2.197	-2.197	1.299	-2.197
9	1.022	0.000	-2.197	-2.197
10	0.000	-0.588	-2.197	0.847
11	0.847	-0.588	-0.588	-0.588
12	0.636	-0.588	0.368	-2.197

Table 3: Position-specific scoring matrices (PSSM) built to study Drosophila genes (http://www.danielpollard.com/matrices.html) of length 12.

Bernoulli models and in more than two times Markov models respectively. In vast majority of cases it outperforms AhoPro in space. The advantage of SufPref the greater the larger space is needed, therefore it can work out the patterns with greater number of words and with greater length.

Availability and requirements

The algorithm SufPref was implemented in a C++ program and was compiled for Unix, Windows and Mac OS. The program was implemented both as web-server and as a stand alone program with the command line interface. It is available at http://server2.lpm.org.ru/bio. Implementation details are provided in http://server2.lpm.org.ru/static/downloads/SufPrefHMM/Web-site.pdf.

Authors contributions

Idea of the work and basic ideas of the algorithms belong to Mireille Régnier, she also took part in the development of the algorithms. Mikhail Roytberg took part in the development of the algorithms especially wrt HMM, he also supervised the programming. Evgenia Furletova took part in the development of the algorithms especially wrt details of implementation and has created the vast majority of the code. Victor Yakovlev consulted Evgenia Furletova during the programming, took part in the program testing and computer experiments and implemented the Web-site.

Acknowledgements

This work was supported by Inria associated team MIGEC and French-Russian grant Carnage.

References

1. Kucherov G, Noé L, Roytberg M: **A unifying framework for seed sensitivity and its application to subset seeds**. *Journal of Bioinformatics and Computational Biology* 2009, **4**(2):553–569.

2. Qian Z, Lu L, Qi L, Li Y: **An efficient method for statistical significance calculation of transcription factor binding sites**. *Bioinformation* 2007, **2**(5):169–174.

3. Berman B, Pfeiffer B, Laverty T, Salzberg S, Rubin G, Eisen M, Celniker S: **Computational identification of developmental enhancers: conservation and function of transcription factor binding-site clusters in Drosophila melanogaster and Drosophila pseudoobscura**. *Genome Biol.* 2004, **5**(9). [R61].

4. Cartharius K, Frech K, Grote K, Klocke B, Haltmeier M, Klingenhoff A, Frisch M, Bayerlein M, Werner T: **MatInspector and beyond: promoter analysis based on transcription factor binding sites**. *Bioinformatics* 2005, :2933–2942.

5. Helden JV, Olmo M, Perez-Ortin J: **Statistical analysis of yeast genomic downstream sequences revels putative polyadenylation signals**. *Nucleic Acids Research* 2000, **28**(4):1000–1010.

6. Roytberg MA: **Computation of the probabilities of families of biological sequences**. *Biophysics* 2009, **54**(5):569–573.

7. Marschal T, Herms I, Kaltenbach H, Rahmann S: **Probabilistic Arithmetic Automata and their Applications**. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2012, **59**(6):1737–1750.

8. Reinert G, Schbath S: **Probabilistic and Statistical Properties of Words: An Overview**. *Journal of Computational Biology* 2000, **7**(1-2):1–46.

9. Nuel G: **Numerical solutions for Patterns Statistics on Markov chains**. *Stat. Appl. Genet. Mol. Biol.* 2006, **5**:26.

10. Lladser M, Betterton MD, Knight R: **Multiple pattern matching: A Markov chain approach**. *Journal of Mathematical Biology* 2008, **56**(1-2):51–92.

11. Guibas L, Odlyzko A: **String Overlaps, Pattern Matching and Nontransitive Games**. *Journal of Combinatorial Theory,* Series A 1981, **30**:183–208.

12. Szpankowski W: *Average Case Analysis of Algorithms on Sequences*. New York: John Wiley and Sons 2001.

13. Régnier M: **A Unified Approach to Word Occurrences Probabilities**. *Discrete Applied Mathematics* 2000, **104**:259–280. [Special issue on Computational Biology;preliminary version at RECOMB'98].

14. Régnier M, Szpankowski W: **On Pattern Frequency Occurrences in a Markovian Sequence**. *Algorithmica* 1997, **22**(4):631–649. [Preliminary draft at ISIT'97].

15. Régnier M, Denise A: **Rare events and Conditional Events on random strings**. *Discrete Mathematics and Theoretical Computer Science* 2004, **6**(2):191–214.

16. Nicodéme P: **Motif statistics**. *Theor. Comput. Sci.* 2004, **287**:593–617.

17. Nicodéme P: **Regexpcount, a symbolic package for counting problems on regular expressions and words**. *Fundamenta Informaticae* 2003, **56**(1-2):71–88.

18. Régnier M, Lifanov A, Makeev V: **Three variations on word counting**. *Proceedings German Conference on Bioinformatics, Heidelberg* 2000, :75–82.

19. Prum B, Rodolphe F, Turckheim E: **Finding words with unexpected frequencies in DNA sequences**. *J. R. Statist. Soc. B* 1995, **11**:190–192.

20. Bender EA, Kochman F: **The Distribution of Subword Counts is Usually Normal**. *Eur. J. Comb.* 1993, **14**(4):265–275.

21. Cowan R: **Expected frequencies of DNA patterns using Whittle's formula**. *J. Appl. Prob.* 1991, **28**:886–892.

22. Godbole AP: **Poissons approximations for runs and patterns of rare events**. *Adv Appl Prob* 1991, **23**:851–865.

23. Geske MX, Godbole AP, Schaffner AA, Skrolnick AM, Wallstrom GL: **Compound Poisson approximations for word patterns under Markovian hypotheses**. *J. Appl. Prob.* 1995, **32**:877–892.

24. Reinert G, Schbath S: **Compound Poisson Approximation for Occurrences of Multiple Words in Markov Chains**. *Journal of Computational Biology* 1998, **5**(2):223–253.

25. Nuel G: **Pattern Markov chains: optimal Markov chain embedding through deterministic finite automata**. *J. Appl. Prob.* 2008, **45**:226–243.

26. L MR, Spouge J, Kanga G, Landsman D: **Statistical analysis of over-represented words in human promoter sequences**. *Nucleic Acids Research* 2004, **32**(3):949–958, [http://0-www.ncbi.nlm.nih.gov.iii-server.ualr.edu/pubmed/14963262].

27. Regnier M, Vandenbogaert M: **Comparison of statistical significance criteria**. *Journal of Bioinformatics and Computational Biology* 2006, **4**(2):537–551.

28. Denise A, Regnier M, Vandenbogaert M: **Assessing the Statistical Significance of Overrepresented Oligonucleotides**. *Lecture Notes in Computer Science* 2001, **2149**:85–97.

29. Nuel G: **LD-SPatt: Large Deviations Statistics for Patterns on Markov Chains**. *J Comp Biol* 2004, **11**(6):1023–1033.

30. Hertzberg L, Zuk O, Getz G, Domany E: **Finding Motifs in Promoter Regions**. *Journal of Computational Biology* 2005, **12**(3):314–330.

31. Boeva V, Clément J, Régnier M, Roytberg M, Makeev V: **Exact p-value calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cis-regulatory modules**. *Algorithms for molecular biology* 2007, **2**(13):25 pages, [http://www.almob.org/content/2/1/13].

32. Nuel G: **Effective p-value computations using Finite Markov Chain Imbedding (FMCI): application to local score and to pattern statistics**. *Algorithms for molecular biology* 2006, **1**(5):14 pages, [http://www.almob.org/content/1/1/5].

33. Zhang J, Jiang B, Li M, Tromp J, Zhang X, Zhang M: **Computing exact p-values for DNA motifs**. *Bioinformatics* 2006, **23**:531–537.

34. Regnier M, Kirakossian Z, Furletova E, Roytberg M A: **A Word Counting Graph**. In *London Algorithmics 2008: Theory and Practice (Texts in Algorithmics)*. Edited by Joseph Chan JWD,

Rahman MS, London: London College Publications 2009:31 p., [http://hal.inria.fr/inria-00437147/en/].

35. Karlin S, Burge C, Campbell A: **Statistical analyses of counts and distributions of restriction sites in DNA sequences**. *Nucleic Acids Research* 1992, **20**(6):1363–1370.

36. Nicodème P, Salvy B, Flajolet P: **Motif Statistics**. *Theoretical Computer Science* 2002, **287**(2):593–618. [Preliminary version at ESA'99].

37. Durbin R, Eddy S, Krogh A, Mitchison G: *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge: Cambridge University 1998.

38. Rabin M: **Probabilistic Automata**. *Information and control* 1963, **6**:230–245.

39. Salomaa A: *Theory of Automata*. Oxford: Pergamon Press 1969. [Chapter 2].

40. Rabiner LR: **A tutorial on hidden Markov models and selected applications in speech recognition**. *Proceedings of the IEEE* 1989, **77**(2):257–286.

41. Aho A, Corasick M: **Efficient String Matching**. *CACM* 1975, **18**(6):333–340.

Figures

Input: an alphabet V, HMM $G =< Q, q_0, \pi >$, the length N of a random text, desired number S of occurrences of pattern words, pattern $\mathcal{H}$
Output: $Prob(B(N,S))$

```
// 1. Pre-processing
Preprocessing;
// 2. Main Loop
foreach n = m+1, ..., N do
    ComputeMainLoop(n);
end
// 3. Post-processing
foreach n = N-m+1, ..., N do
    foreach q ∈ Q do
        Compute Prob(B(n,S),q) by the formula (14);
    end
end
Compute Prob(B(N,S)) by summation of the values Prob(B(N,S),q);
```

Figure 1. Algorithm $SufPref$.

Input: **integer** n, $n > m$

// Preliminary step. Initialization $EProb()[]$ and $EProbPrev()[]$
foreach $r \in DROV(\mathcal{H})$ **do**
  foreach $q \in AllState(r)$ **do**
    Compute $Prob(E(n,1,r),q)$ according (15) and set the value to $EProbPrev(r)[q]$;
    $EProb(r)[q] = 0$
  end
end
foreach $s = 1, \ldots, S$ **do**
  // A. Computation of $Prob(B(n-m,s),q)$
  if $n > 2m$ **then**
    foreach $q \in Q$ **do**
      Compute $Prob(B(n-m,s),q)$ using (14);
    end
  end
  // B. Computation of $Prob(E(n,s,\mathrm{r}),q)$ for all right deep nodes r
  foreach *node w visited within the depth-first traversal of OvGraph by the left edges* **do**
    foreach $q \in AllState(w)$ **do**
      Compute $Prob(D(k(n,w),s,w),q)$ using (13);
    end
    if *w is left deep node* **then**
      foreach *deep edge* (w,r) **do**
        foreach $q \in AllState(\mathrm{r})$ **do**
          Compute $Prob(C(n,s+1,w,\mathrm{r}),q)$ using (18);
          $EProb(\mathrm{r})[q] += Prob(C(n,s+1,w,\mathrm{r}),q)$;
        end
      end
    end
    if *w is right deep node* **then**
      foreach $q \in AllState(w)$ **do**
        Compute $Prob(F(n,s+1,w),q)$ using (17);
        $EProb(w)[q] += Prob(F(n,s+1,w),q)$;
      end
    end
  end
  // C. Computation of $Prob(R(n,s,w),q)$ for all nodes w
  foreach *node w of OvGraph traversed from deep right nodes to the root* **do**
    foreach $q \in AllState(w)$ **do**
      $REProb = 0$;
      if *w is right deep node* **then**
        // Compute $Prob(RE(n,s,w),q)$ based on arrays $EProb(), EProbPrev()$ and update the arrays
        $REProb = EProbPrev(w)[q] - EProb(w)[q]$;
        $EProbPrev(w)[q] = EProb(w)[q]$;
        $EProb(w)[q] = 0$;
      end
      Compute $Prob(R(n,s,w),q)$ following (8) and (9);
    end
  end
end

Figure 2. Sub-algorithm $ComputeMainLoop$.